MySQL Database Syntax Explanation

Whenever using MySQL in the future, start by entering the command mysqlsh --**mysql -u root -p** in the command prompt (cmd) to switch to the MySQL environment. After generating the Excel file, you will need to import the remaining tables into the database.

CREATE DATABASE database_name; - This command creates a database.

USE database_name; - Use this command to select the database.

SHOW DATABASES; - View all databases.

DROP DATABASE database_name; - Delete a specific database.

Data Types

In addition to creating a database, you can also create tables with similar properties. Data types are similar to those in programming, such as int and string.

INT: Integer type

VARCHAR: Variable-length string

TEXT: Text type

DATE: Date type

DATETIME: Date and time type

FLOAT: Floating-point type

DOUBLE: Double-precision floating-point type

BOOLEAN: Boolean type, storing TRUE or FALSE

Constraints: **Constraints ensure the integrity and correctness of the data.**

PRIMARY KEY: Uniquely identifies each row in the table.

FOREIGN KEY: Foreign key constraint.

UNIQUE: Ensures the uniqueness of values in a column.

NOT NULL: Ensures that a column cannot have NULL values.

DEFAULT: Provides a default value for a column.

AUTO_INCREMENT: Automatically increments the value.

Example:

```
CREATE DATABASE sensor_data;

USE sensor_data;

CREATE TABLE Sensors (
    Sensor_Key INT PRIMARY KEY,
    Sensor_ID VARCHAR(255) NOT NULL,
    Location VARCHAR(255) NOT NULL
);


CREATE TABLE Measurements (
    ID INT PRIMARY KEY NOT NULL,
    Temperature FLOAT NOT NULL,
    Humidity INT NOT NULL,
    Timestamp DATETIME NOT NULL,
    TimeZone VARCHAR(255) NOT NULL,
    Temperature_Change_Rate VARCHAR(255),
    Humidity_Change_Rate VARCHAR(255),
    Comfort_Index FLOAT,
    Location VARCHAR(255) NOT NULL,
    Sensor_ID INT NOT NULL,
    Comfort_Level VARCHAR(255),
    FOREIGN KEY (Sensor_ID) REFERENCES Sensors(Sensor_Key)
);
```

This example creates a database, uses the database, and creates a table within the database. The Sensors table contains three columns. The example includes creating a Sensors table and a Measurements table. The last line adds a foreign key constraint. Sensor_Key is the primary key in the Sensors table, and Sensor_ID is used as a foreign key in the Measurements table, referencing the primary key in the Sensors table. This establishes a relationship between the two tables.

Part Two: Inserting Data into MySQL

We first create an empty table, then use a method to insert all Excel data into MySQL. The necessary libraries are mysql-connector-python and pandas. The Pandas library is commonly used, and mysql-connector-python is used for connecting MySQL with Python, as its name suggests.

Install these libraries in the terminal using **pip install mysql-connector-python.**

```
import pandas as pd
import mysql.connector
import numpy as np

# Connect to the MySQL database
connection = mysql.connector.connect(
    host='localhost',
    user='root',
    password='123Wjb456+',
    database='sensor_data'
)

cursor = connection.cursor() # Create a cursor object to execute SQL statements

file_path = 'C:/Users/Frank/Desktop/New_DHT11_Data_Collection.xlsx' # Read data
from the Excel file
sensors = pd.read_excel(file_path, sheet_name='Sensors') # Read the 'Sensors'
sheet
measurements = pd.read_excel(file_path, sheet_name='Measurements') # Read the
'Measurements' sheet

# Remove duplicate primary key records
sensors = sensors.drop_duplicates(subset=['Sensor Key'])
measurements = measurements.drop_duplicates(subset=['ID'])

try:
    for index, row in sensors.iterrows(): # Iterate through each row of the
Sensors DataFrame and insert data into the Sensors table, ignoring duplicate
primary keys
        sql = "INSERT IGNORE INTO Sensors (Sensor_ID, Location, Sensor_Key)
VALUES (%s, %s, %s)" # Define the insert statement, using INSERT IGNORE to avoid
duplicate primary key errors
        params = (row['Sensor ID'], row['Location'], row['Sensor Key']) # Extract
data from the current row
        print(f"Executing SQL: {sql} with params {params}") # Print the SQL
statement and parameters to be executed for debugging purposes
```

```
        cursor.execute(sql, params) # Execute the insert operation

    for index, row in measurements.iterrows(): # Insert data into the
Measurements table, ignoring duplicate primary keys
        sql = """INSERT IGNORE INTO Measurements (ID, Temperature, Humidity,
Timestamp, TimeZone, Temperature_Change_Rate,
                  Humidity_Change_Rate, Comfort_Index, Location, Sensor_ID,
Comfort_Level)
                  VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)""" #
Define the insert statement, using INSERT IGNORE to avoid duplicate primary key
errors

        params = (row['ID'], row['Temperature'], row['Humidity'],
row['Timestamp'], row['TimeZone'],
                  row['Temperature Change Rate'], row['Humidity Change Rate'],
row['Comfort Index'], row['Location'], row['Sensor ID'], row['Comfort Level']) #
Extract data from the current row
        print(f"Executing SQL: {sql} with params {params}") # Print the SQL
statement and parameters to be executed for debugging purposes
        cursor.execute(sql, params) # Execute the insert operation

    connection.commit() # Commit changes to the database to ensure all insert
operations are saved

except mysql.connector.Error as err: # Catch and print database operation errors
    print("Error: {}".format(err))

finally:
    cursor.close() # Close the cursor
    connection.close() # Close the connection
```

Explanation of the code segment by segment:

```
# Connect to the MySQL database
connection = mysql.connector.connect(
    host='localhost',
    user='root',
    password='123Wjb456+',
    database='sensor_data'
)
```

In the first half, use mysql.connector to establish a connection. Remember that the host is usually a local host, user can be root or a user you created yourself. If you want to call it, the

password is the mysql password, database is created, and python is used to establish a connection.

```
file_path = 'C:/Users/Frank/Desktop/New_DHT11_Data_Collection.xlsx' # Read data
from the Excel file
sensors = pd.read_excel(file_path, sheet_name='Sensors') # Read the 'Sensors'
sheet
measurements = pd.read_excel(file_path, sheet_name='Measurements') # Read the
'Measurements' sheet

# Remove duplicate primary key records
sensors = sensors.drop_duplicates(subset=['Sensor Key'])
measurements = measurements.drop_duplicates(subset=['ID'])
```

Then the following part of the file_path uses the latest updated Excel. Then it reads two different tables in the same Excel. One is Sensors and the other is Measurements, which also corresponds to our MySQL database. The next step is to drop any duplicate primary keys.

In a database**, the primary key is used to uniquely identify each row of records. If there are duplicate primary keys, data integrity will be destroyed, leading to confusion in data query, update, and delete operations**, making it impossible to accurately locate specific records, and **may cause errors or even crashes in applications that rely on the uniqueness of the primary key.** Therefore, when someone gives you a data table with a large amount of data and you can't check it one by one, just use this method to delete the duplicates.

```
cursor = connection.cursor() # Create a cursor object to execute SQL statements


try:
    for index, row in sensors.iterrows(): # Iterate through each row of the
Sensors DataFrame and insert data into the Sensors table, ignoring duplicate
primary keys
        sql = "INSERT IGNORE INTO Sensors (Sensor_ID, Location, Sensor_Key)
VALUES (%s, %s, %s)" # Define the insert statement, using INSERT IGNORE to avoid
duplicate primary key errors
        params = (row['Sensor ID'], row['Location'], row['Sensor Key']) # Extract
data from the current row
        print(f"Executing SQL: {sql} with params {params}") # Print the SQL
statement and parameters to be executed for debugging purposes
        cursor.execute(sql, params) # Execute the insert operation
```

```
    for index, row in measurements.iterrows(): # Insert data into the
Measurements table, ignoring duplicate primary keys
        sql = """INSERT IGNORE INTO Measurements (ID, Temperature, Humidity,
Timestamp, TimeZone, Temperature_Change_Rate,
                    Humidity_Change_Rate, Comfort_Index, Location, Sensor_ID,
Comfort_Level)
                    VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)""" #
Define the insert statement, using INSERT IGNORE to avoid duplicate primary key
errors

        params = (row['ID'], row['Temperature'], row['Humidity'],
row['Timestamp'], row['TimeZone'],
                  row['Temperature Change Rate'], row['Humidity Change Rate'],
row['Comfort Index'], row['Location'], row['Sensor ID'], row['Comfort Level']) #
Extract data from the current row
        print(f"Executing SQL: {sql} with params {params}") # Print the SQL
statement and parameters to be executed for debugging purposes
        cursor.execute(sql, params) # Execute the insert operation
```

Use the try block to catch and handle possible database operation exceptions.

Sensors table / Measurements table:

- Traverse each row of data, define the insert statement and extract the parameters from the current row.
- Use the INSERT IGNORE statement to ignore the insert operation if the primary key is repeated.
- Print the SQL statement and parameters to be executed for easy debugging.
- Execute the insert operation.
- Commit the changes using **connection.commit()** to ensure that all insert operations are saved to the database.

The second part of the code starts with the cursor, which is the location of each item, similar to the point where a cursor is located. Then the for loop starts, traversing the rows in Excel one by one, each item is processed and then inserted into the Sensors table in the database. %s is a placeholder and has no actual meaning. The actual parameter is passed later.

```
    connection.commit() # Commit changes to the database to ensure all insert
operations are saved

except mysql.connector.Error as err: # Catch and print database operation errors
    print("Error: {}".format(err))
```

```
finally:
    cursor.close() # Close the cursor
    connection.close() # Close the connection
```

The first of these three lines is connection commit, and all three lines are written in a fixed way. **The first line is commit, and the last two lines are close.**

To summarize, **the role of cursor is responsible for managing the cursor in the database connection, allowing the records in the database to be processed row by row.** Let's go through the code. First, connect to the MySQL database using mysql.connector.connect. Then create a cursor object and read the excel file data. The reading part uses the old pandas method. Then insert the sensor data, using the method of traversing each data one by one. Finally, submit the data and close the data.

output